REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*
9 April 2010

2. REPORT TYPE
Journal Article

3. DATES COVERED *(From - To)*
June 2009-June 2010

4. TITLE AND SUBTITLE
CUDA Implementation of TE^{z-}FDTD Solution of Maxwell's Equations of Dispersive Media

5a. CONTRACT NUMBER
FA8650-07-C-6756

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
63231F

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
5020D481

6. AUTHOR(S)
Mohammad R. Zunoibi, Jason Payne, and William P Roa

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

711 HPW/RHDR
8262 Hawks Road
Brooks City-Base, TX 78235

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Materiel Command
Air Force Research Laboratory
Human Effectiveness Directorate
Directed Energy Bioeffects Division
Radio Frequency Radiation Branch
8262 Hawks Road
Brooks City-Base, TX 78235

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL
711 HPW/RHD

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-RH-BR-JA-2010-0013

12. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution A. Approved for public release. Distribution unlimited. Public Affairs Case File No. 10-270. 15 July 2010. Published Journal Article

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This research presents the Graphic Processor Unit (GPU) implementation of the Finite-Difference Time-Domain (FDTD) method for the solution of the 2-dimensional electromagnetic fields inside dispersive media. The FDTD domain is truncated by the convolutional perfectly matched layer (CPML) and the Piecewise-Linear Recursive-Convolution (PLRC) formulation is used for modeling dispersive media. By using the newly introduced CUDA technology, we illustrate the efficacy of GPUs in accelerating the FDTD computations by achieving significant speedup factors with great ease and at no extra hardware/software cost. We validate our approach by comparing with exact and other simulated results, which show favorable agreements. The effect of the GPU-CPU memory transfers on the speedup factor will be also studied.

15. SUBJECT TERMS
Convolutional perfectly matched layer (CPML), dispersive, Finite-Difference Time-Domain (FDTD), **Graphic Processor Unit** (GPU)

16. SECURITY CLASSIFICATION OF:

a. REPORT	b. ABSTRACT	c. THIS PAGE	17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UU	4	19b. TELEPHONE NUMBER *(include area code)*

CUDA Implementation of TE^z-FDTD Solution of Maxwell's Equations in Dispersive Media

Mohammad Reza Zunoubi, *Senior Member, IEEE*, Jason Payne, and William P. Roach

***Abstract*—This letter presents the graphic processor unit (GPU) implementation of the finite-difference time domain (FDTD) method for the solution of the two-dimensional electromagnetic fields inside dispersive media. The FDTD is truncated by the convolutional perfectly matched layer (CPML) and the piecewise-linear recursive-convolution (PLRC) formulation is used for modeling dispersive media. By using the newly introduced Compute Unified Device Architecture (CUDA) technology, we illustrate the efficacy of GPUs in accelerating the FDTD computations by achieving significant speedup factors with great ease and at no extra hardware/software cost. We validate our approach by comparison to exact and other simulated results, which show favorable agreements. The effect of the GPU–CPU memory transfers on the speedup factor will be also studied.**

***Index Terms*—Convolutional perfectly matched layer (CPML), dispersive, finite-difference time domain (FDTD), graphic processor unit (GPU).**

I. Introduction

The interaction of electromagnetic fields with human tissues and the resulting biological effects have been of great interest to both commercial and government sectors. The finite-difference time domain (FDTD) technique has been a method of choice for many years in studying this interaction. FDTD's popularity is partly due to its straightforward implementation and its ability to model a broad range of exposure conditions for anisotropic, nonlinear, and dispersive media. However, the FDTD algorithm can be computationally expensive, and significant run-times may exist for certain applications.

Recently, NVIDIA has introduced a general-purpose parallel computing architecture called Compute Unified Device Architecture (CUDA) that leverages the parallel compute engine in NVIDIA graphic processor units (GPUs) to solve many complex computational problems in a fraction of the time required on a CPU [1]. Through CUDA, the vast variety of GPU memory hierarchy may be accessed, which allows for maximum optimization of computational schemes that are parallel in nature, such as FDTD.

As commercial sectors like Remcom, Computer Simulation Technology (CST), Acceleware, Schmid & Partner Engineering AG (SPEAG), and Agilent have integrated the GPU-accelerated FDTD technique into their software packages to treat electromagnetic problems in general dispersive and nondispersive media, researchers in academia have also focused on taking advantage of inherent attributes of the GPUs for parallel computing. Such implementations that use the CUDA technology include the works presented in [2] and [3] originally and the research presented by Sypek *et al.* for a TM^z solution of electromagnetic fields [4], the double-precision implementation of the FDTD on the Tesla GPU [5], and the work reported in [6] more recently, although earlier successful FDTD GPU-accelerated methodologies using less versatile APIs such as Brook or OpenGL were reported starting a few years ago.

Here, we present the implementation of the two-dimensional (2-D) FDTD method in dispersive media on GPUs using CUDA technology. The convolutional perfectly matched layer (CPML) is used to truncate the solution domain [7], and a plane-wave incident field is utilized to simulate realistic exposure scenarios. The software validity is established by comparison to exact results for the reflection coefficient of a dispersive half-space and other simulated results for more complex geometries. Speedup factors are reported that show promise for the rapid solution of large-scale electromagnetic radiation and scattering problems. We also study the effects of the memory transfer between the GPU and CPU on the speedup factor.

II. FDTD Formulation

In the following, we present the analysis of TE^z-polarized plane wave penetration through two-dimensional dispersive media. As such, we follow the procedure outlined in [7] for the piecewise-linear recursive-convolution (PLRC) method for the single-pole Debye representation of the dispersive media along with the CPML boundary condition, which is not only efficient in minimizing the memory requirements, but also the most accurate form of absorbing material. This, in turn, yields the following form of the FDTD update equation for the x-component of the electric field:

$$\begin{aligned} E_x\big|_{i+1/2,j}^{n+1} &= c_a\big|_{i+1/2,j} E_x\big|_{i+1/2,j}^{n} + c_b\big|_{i+1/2,j} \\ &\quad \cdot \left(\frac{H_z\big|_{i+1/2,j+1/2}^{n} - H_z\big|_{i+1/2,j-1/2}^{n}}{\kappa_{yj}} + \psi_{E_{x,y}}\big|_{i+1/2,j}^{n} \right) \\ &\quad + c_c\big|_{i+1/2,j} \psi_{E_x}\big|_{i+1/2,j}^{n} \end{aligned} \tag{1}$$

where C_a, C_b, and C_c are the coefficients associated with space–time discretization and material properties, κ_{yj} is the scaled tensor parameter, $\psi_{E_{x,y}}$ is the auxiliary array for the CPML, and ψ_{E_x} is the recursive accumulator variable given

Manuscript received March 18, 2010; revised May 24, 2010 and June 22, 2010; accepted July 07, 2010. Date of publication July 19, 2010; date of current version August 09, 2010.

M. R. Zunoubi is with the Electrical and Computer Engineering Department, State University of New York, New Paltz, NY 12561 USA (e-mail: zunoubm@engr.newpaltz.edu).

J. Payne is with the Human Effectiveness Directorate (AFRL/HE), U.S. Air Force Research Laboratory, Brooks City-Base, San Antonio, TX 78235 USA.

W. P. Roach is with the Advanced Electric Laser Branch (AFRL/RDLA), Kirtland Air Force Base, NM 87117 USA.

Digital Object Identifier 10.1109/LAWP.2010.2060181

1536-1225/$26.00 © 2010 IEEE

in [8] and defined in terms of ε_s, ε_∞, and τ, which are the static relative permittivity, the relative permittivity at infinite frequency, and relaxation time, respectively, for the Debye material.

To model the tissue material, we generate an *id* file that is used to assign the constitutive parameters to the corresponding field components. For *normal* materials, and ε_r, μ_r and σ are assigned and used to calculate the regular FDTD updating coefficients. For *dispersive* materials, the Debye parameters $\varepsilon_\infty, \varepsilon_s$, and τ are assigned and used to calculate the dispersive FDTD updating coefficients for the electric fields and for the recursive accumulator variable ψ_{E_x}.

III. CUDA Implementation

A. Memory Management

Since we are implementing a TE^z formulation, we allocate and initialize on the *global* memory of the *device* (GPU) three one-dimensional arrays for the E_x, E_y, and H_z components of the fields, four auxiliary ψ-arrays for the CPML such as $\psi_{E_{x,y}}$, two arrays for recursive accumulator variables ψ_{E_x} and ψ_{E_y}, and two arrays for the incident electric and magnetic field components. Note that all these arrays will have to be updated during the FDTD time-marching process. However, once on the *device*, we will use the fast but limited *shared* memory as described in the next section to handle updating equations. Additionally, we need to allocate and assign two integer arrays *tissueIDx* and *tissueIDy* that contain the indices assigned to the different tissue material under exposure, and 16 arrays for coefficients c_a, c_b, and c_c, scaled tensor parameters κ's, and the constants used in calculating auxiliary ψ-arrays. However, since no updating of these arrays is needed during the time-marching process, we use the *texture* memory of the *device* for storage, which allows for fast read-access when updating the field quantities.

B. Device Kernels

For efficient implementation of our FDTD approach, we divided the *device* implementation into two *kernels*: one to update the magnetic field component and another to update the electric field intensities. To take full advantage of the single-instruction multiple-date (SIMD) architecture of the GPU and overcome the memory latency, we group the threads into $i \times j$ 2-D thread blocks. One thread is assigned to an output element while allowing four halo regions to include the neighboring elements that are needed in the FDTD updating equations. This method allows the threads within each small block to access their own *shared* memory, the latency of which is two orders of magnitude lower than that of *global* memory. Therefore, during each time step, the field and ψ-arrays are loaded into the *shared* memory, the updating equations are operated upon, and the resulting updated electric and magnetic fields are stored in the *global* memory to be read by the *host*.

Note that the thread dimensions i and j of each block need to be optimized in order to take full advantage of the GPU computational capabilities. This will be addressed in the Section IV.

Following, we illustrate our CUDA implementation by describing the kernels on the *device* side that use the *texture*, *shared*, and *global* memory hierarchies to perform the FDTD updating equations efficiently. Note that before invoking these kernels, the main program on the *host* side should manage the *device* memory allocations, copying data from *host* to *device* and copying the data after FDTD updating (invocation of kernels) from the *device* to the *host*:

- Load the read-only data in the *texture* memory:
  texture<int, 1, cudaReadModeElementType>ttissueIDx;
- Repeat for all other constant arrays;
- update H<Grid of thread blocks, Size of thread block>
  {
    Compute spatial coordinate of current thread
    Load Ex and Ey from the *global* memory into the *shared* memory to avoid GPU's memory latency;
    Perform FDTD updating equations for H while reading constant arrays from the fast *texture* memory;
    Copy data from the *device global* memory back to the *host*;
  }
- update E<Grid of thread blocks, Size of thread block>
  {
    Compute spatial coordinate of current thread
    Load Hz from the *global* memory into the *shared* memory to avoid GPU's memory latency;
    Perform FDTD updating equations for E while reading constant arrays from the fast *texture* memory:
    — Compute CPML modifiers (d_PsiExy, d_PsiEyx)
    — Perform updating of E-fields including the CPML for no-dispersive regions
    — Update ψ_{E_x} and ψ_{E_y} matrices for dispersive material
    — Update the E fields in dispersive regions
    — Correct for plane wave interface
    Copy data from the *device global* memory back to the *host*;
  }

C. Arithmetic Instructions

To maximize the computational efficiency of our *kernels*, we use the $\_\_mul24$ and $\_\_fdividef$ functions for integer multiplications and floating-point divisions, respectively. $\_\_mul24$ provides 24-bit integer multiplication in four clock cycles compared to the 16 clock cycles for the conventional 32-bit multiplication, and $\_\_fdividef$ performs single-precision floating-point division in 20 clock cycles, which is superior to the 36 clock cycles typically needed for dividing floating point values [1].

IV. Results

Before studying the advantages of the CUDA implementation of the FDTD for dispersive media, we demonstrate the accuracy of our developed tool by first analyzing the problem of a Gaussian plane wave propagating in vacuum normally incident on a vacuum–water interface. The problem space is divided into 1024 cells in both x- and y-directions with a cell size of 37.5 μm, which ensures the accuracy of the results up to 400 GHz in free-space regions based on the 20-cells/λ criterion. The time step is chosen according to the Courant limit as $\Delta t = 0.0884$ ps, and we use an eight-cell CPML absorbing boundary to truncate the volume. To model the frequency dependence of water, the Debye parameters of $\varepsilon_{\mathrm{S}} = 81$, $\varepsilon_\infty = 1.8$, $\sigma_{\mathrm{S}} = 0$, and $\tau = 9.4\mathrm{e}-12$ are used. The FDTD calculation is performed

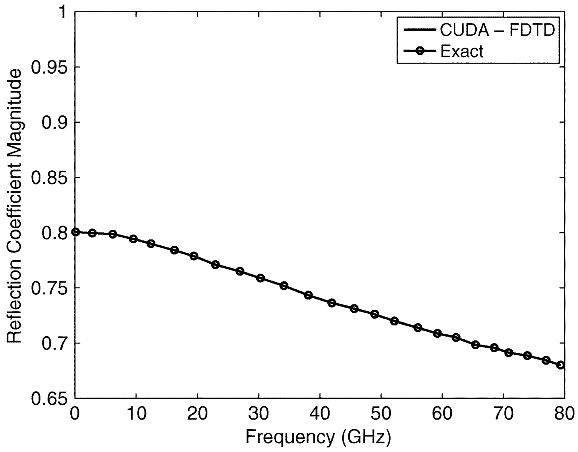

Fig. 1. The magnitude of the reflection coefficient calculated at the vacuum-water interface.

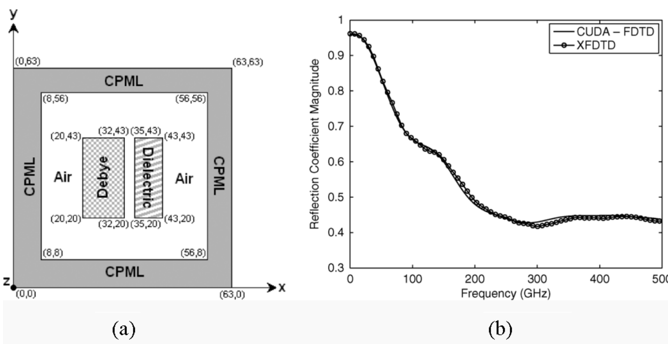

Fig. 2. (a) Problem geometry for a combination of dispersive and nondispersive materials. (b) Magnitude of the reflection coefficient calculated at the vacuum–Debye interface.

for 1300 time steps, and results are recorded at the cell location (504, 504), which is on the vacuum–water boundary. The incident field due to the Gaussian pulse, which has duration of 256 time steps, is subtracted from the total field E_y to obtain the reflected field due to the material discontinuity. The reflection coefficient versus frequency is then calculated by normalizing the Fourier transform of the reflected field by the Fourier transform of the incident pulse. Results can be seen in Fig. 1, where a comparison is made with the exact solution, illustrating the adequacy and accuracy of our FDTD-CUDA implementation.

To further demonstrate the accuracy of our implementation, we consider a 2-D region shown in Fig. 2(a), which consists of layers of vacuum, a Debye slab, vacuum, a dielectric slab, and vacuum again, and we compute the reflection coefficient as discussed on the vacuum–Debye material interface versus frequency. Note that all the dimensions are in terms of the FDTD voxels. The Debye material properties are for water as indicated previously, and the dielectric slab has $\varepsilon_r = 2$. Fig. 2(b) shows the results along with comparisons to the corresponding results obtained from the Remcom XFDTD software, where a very good agreement is observed. Note that for the XFDTD results, the region is extended in the z-direction by 400 FDTD cells to mimic a 2-D problem. In our computation, the same spatial discretization as the above example was chosen to assure the accuracy of the results up to 400 GHz while the simulation was performed for 1100 time steps, which insured achieving the steady-state condition.

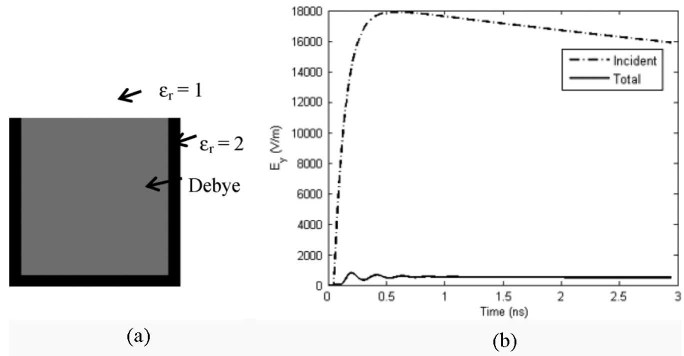

Fig. 3. (a) Cuvette filled with water modeled as Debye material. (b) Incident and total electric field calculated at the center of the cuvette.

We next demonstrate the applicability of our tool to calculating the absorbed dose of dispersive media exposed to nonionizing ultrawideband (UWB) electromagnetic pulses of nanosecond duration. This dosimetry is critical for establishing dose-response curves for nanosecond electromagnetic pulses. Here, we consider the problem of exposing water inside a cuvette to a short pulse described as [9]

$$E_y = E_0(e^{-\alpha t} - e^{-\beta t}) \qquad \text{with } E_0 = 18.5\,\mathrm{KVm^{-1}}, \\ \alpha = 10^8\,\mathrm{s^{-1}}, \quad \beta = 2.0 \times 10^{10}\,\mathrm{s^{-1}} \quad (2)$$

which has a rise time of 150 ps and a width of 10 ns. The cuvette is 1×1 cm^2 with 1-mm-thick plastic walls, as shown in Fig. 3(a), and is filled with water that has the dispersive properties described. Since the penetration of the electric field component in the direction of polarization is defined by the rise time and pulsewidth, we plot both the incident field and the total field inside water in the middle of the cuvette in Fig. 3(b). This plot indicates that the pulse inside water is a superposition of a short pulse, induced during a fast rise time, and recursively longer pulses induced by the slow variation in the incident pulse. This is similar to the observation made in [9].

As discussed in the previous section, the size of the thread blocks loaded into *shared* memory could have a significant effect on the performance of the GPUs. Accordingly, a study was performed to identify the optimum block dimension for our FDTD computations. We executed the CUDA program for 2000 time steps for various thread arrangements following the format of $(i \times j) = (2 \times j), (4 \times j), (8 \times j), (16 \times j), (32 \times j), (64 \times j), (128 \times j)$, and $(256 \times j)$ while setting $j = 2$, 4, 8, 16, 32, 64, 128, and 256 with a problem space of size 1024×1024 FDTD cells. This study indicated that a block size of $i \times j = 16 \times 4$, which results in 64 threads per block and uses 256 bytes of the available 16 kB of memory, yields the optimum performance with a GPU computation time of 7.87 s.

Having identified the optimum block dimension, we next investigate the speedup factors achieved using the GPU as we increase our FDTD domain size from 128×128 cells (16 384 grid points) to 2048×2048 cells (4 194 304 grid points). The FDTD computations are performed for 2000 time steps on both C and CUDA versions of our code. The C implementation was performed on Intel Core2 Quad Q6600, 2.40 GHz, and 3.25 GB of RAM. The CUDA tool was run on an NVIDIA GeForce 9800 GT, which has 112 stream processors, 1024 MB standard memory, a memory bandwidth of 57.6 GB/s, and a shader processing rate of 504 gigaflops. The computation times for both

TABLE I
CPU AND GPU FDTD COMPUTATION TIME

Grid points	CPU (s)	GPU (s)	Speedup
16,384	9.28	0.39	23.8
65,536	43.81	0.76	57.6
262,14 4	186.4	2.20	84.72
1,048,576	794.5	8.05	98.70
4,194,304	3114.7	31.6	98.56

C and CUDA implementations as well as the achieved speedup factors are given in Table I. These results show a significant improvement in the FDTD calculations time for the GPU versus the CPU implementation, and it is seen that speedup factor of 98.56 can be obtained for up to 4 194 304 grid points. It is expected, however, that with the newly manufactured NVIDIA Tesla C1060 GPUs, these computations can be carried out for hundreds of millions of FDTD grid points with even greater computational efficiency. It is also to be noted that the time taken to transfer the data from the *host* to the *device* and vice versa was negligible and was found to be only 0.17 s for the 4 194 304 FDTD cells when transferring the initial values of the fields and material arrays to the *device*.

It is known that for some practical applications, there is a need for recording the time-domain field values at several locations inside the FDTD solution domain during the time-marching process. When working with the GPU *kernels*, there are three possible approaches. One is to store such data on the *device* as one iterates through all time steps and transfer the data to the *host* at the end of the time-marching process. This would obviously have less effect on the speedup factors that can be achieved by performing the computations on the GPU. However, this would require allocating and using part of the *global* memory of the *device*, further limiting the number of unknowns that can be treated using one single GPU. The second option would be to save the sample points on the *device* only during each single time step and transfer them to the *host* after the updating equations are performed to then be saved in their corresponding arrays on the *host*. This approach needs a small amount of the *device* memory usage and should not affect the speedup factors considerably. The third method would be to transfer the entire array of the desired field component(s) to the *host* after every time step, and then save the required sample point locations in their corresponding arrays on the *host*, thus eliminating any need for additional storage on the *device*. By eliminating the memory requirements on the *device*, all the *device* memory space can be devoted to performing the FDTD updating equations at the cost of lowering speedup factors due to the memory transfer at each time step from the *device* to the *host* specially as the problem size grows.

In order to investigate the data-transfer effect, we studied the effect of sampling the y-component of the electric field at 10 equally spaced locations along the x-axis at the middle of the FDTD solution domain for 8192 time steps, first by computing/saving the sampled field values on the *device* for each time step and then transferring the data onto the *host*, and second, by transferring the whole array of the y-component of the electric field after each time step and storing only the sample points results on the *host*. As expected, for the first approach, no appreciable reduction in the speedup factors was observed, while some *device* memory needed to be allocated for the storage of the sample points. Following the second method reduced the speedup factors from 98.7 and 98.56 s for 1 048 576 and 4 194 304 unknowns to 39.5 and 43.5 s, respectively, further demonstrating the fact that transferring large arrays of field components from the *device* to the *host* can reduce the speedup factors considerably.

V. CONCLUSION

We presented a CUDA-based FDTD analysis of dispersive media on an NVIDIA 9800 GT GPU card. The accuracy of our implementation was illustrated by comparing our results with their corresponding analytical and simulated results. Very good agreement was observed. We further investigated the computational advantage of performing the FDTD equations on GPUs by solving for the electromagnetic fields in dispersive media for a problem space of 2048×2048 voxels. It was shown that speedup factors of up to 98.56 can be achieved by facilitating the *shared* memory of the thread blocks and by using the optimized block dimensions and arithmetic of the CUDA technology. The effects of memory transfer between the device and the host were also studied for 10 sample locations, illustrating the fact that if large arrays of data are to be transferred from the *device* to the *host* at each time step, considerable reduction in speedup factors may occur. We are currently implementing the more practical three-dimensional (3-D) version of our tool and expect to obtain considerable speedup factors with minimal or no additional software/hardware resources, although it is known that the 3-D implementation would not yield large speedup factors achieved with the 2-D tool since the GPU structure is optimally made for the support of the 2-D images.

ACKNOWLEDGMENT

M. R. Zunoubi thanks Dr. A. Elsherbeni from the University of Mississippi, Oxford, for introducing him to the GPU computing idea and Mr. A. Valcarce of the University of Bedfordshire, Luton, U.K., for his help with understanding CUDA structure.

REFERENCES

[1] *"NVIDIA CUDA Compute Unified Device Architecture Programming Guide,"* ver. 2.3, NVIDIA Corporation, Jul. 2009.

[2] A. Balevic, L. Rockstroh, A. Tausendfreund, S. Patzelt, G. Goch, and S. Simon, "Accelerating simulations of light scattering based on finite-difference time-domain method with general purpose GPUs," in *Proc. 11th IEEE CSE*, São Paulo, Brazil, Jul. 2008, pp. 16–18.

[3] A. Valcarce, G. De La Roche, A. Juttner, D. López-Pérez, and J. Zhang, "Applying FDTD to the coverage prediction of WiMAX femtocells," *Eurasip J. Wireless Commun. Netw.*, vol. 2009, Feb. 2009, Article no. 3.

[4] P. Sypek, A. Dziekonski, and M. Mrozowski, "How to render FDTD computations more effective using a graphics accelerator," *IEEE Trans. Magn.*, vol. 45, no. 3, pp. 1324–1327, Mar. 2009.

[5] V. Demir, "Performance analysis of CUDA implementation of FDTD on Tesla GPU using double precision arithmetic," in *Proc. 2010 USNC-URSI Nat. Radio Sci. Meeting*, Boulder, CO, Jan. 6–9, 2010.

[6] C. Y. Ong, M. Weldon, S. Quiring, L. Maxwell, M. C. Hughes, C. Whelan, and M. Okoniewski, "Speed it up," *IEEE Microw. Mag.*, vol. 11, no. 2, pp. 70–78, Apr. 2010.

[7] J. A. Roden and S. D. Gedney, "Convolutional PML (CPML): An efficient FDTD implementation of the CFS-PML for arbitrary media," *Microw. Opt. Technol. Letters*, vol. 27, pp. 334–339, Jun. 2000.

[8] A. Taflove and S. C. Hagness, *Computational Electrodynamics: The Finite-Difference Time-Domain Method*, 3rd ed. Norwood, MA: Artech House, 2005.

[9] N. Simicevic and D. T. Haynie, "FDTD simulation of exposure of biological material to electromagnetic nanopulses," *Phys. Med. Biol.*, vol. 50, no. 2, pp. 347–360, 2005.